Computers and Mathematics with Applications 59 (2010) 2951–2963

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

MAPLE code for the gamma algorithm for global optimization of uncertain functions in economy and finance

M. Delgado Pineda [a], E.A. Galperin [b,*]

[a] Departamento de Matemáticas Fundamentales, Universidad de Educación a Distancia, Facultad de Ciencias, c/ Senda del Rey 9, C.P. 28040 Madrid, Spain

[b] Département de Mathématiques, Université du Québec à Montréal, C.P. 8888, Succ. Centre Ville, Montréal, Québec H3C 3P8, Canada

ARTICLE INFO

Article history:
Received 10 February 2010
Accepted 10 February 2010

Keywords:
Global optimization of uncertain functions
Gamma algorithm
Cubic algorithm

ABSTRACT

Problems with uncertainties are ubiquitous in many areas of life and the economy. Due to a lack of information as regards the economy and in finance, problems with uncertainties (stock prices, marketing problems, inflation, unemployment) are usually formulated by giving bounds on maximum and minimum values of certain parameters, i.e. box constraints. In such situations, it is necessary to make a choice of better parameters that produce finite intervals of possible values for a given uncertain function at each point of the parameter space. The gamma algorithm presents a method for making that choice. A variant of the gamma algorithm based on the cubic algorithm is considered, for global optimization of uncertain functions with box constraints in $\mathbb{R}^n$. The set-monotonic algorithm contains a block for problems with equality constraints, and operates within the unit cube $[0, 1]^n$ for all problems. On this basis, a MAPLE code of modular structure is developed for full global optimization of uncertain functions in n variables. The code does not create ill-conditioned situations. Graphics are included, and the solution set can be visualized in plane projections and sections. An example related to Minsky's Financial Instability Hypothesis is presented, with a graph, to illustrate the use of the code.

1. Introduction

An uncertain function f can be interpreted via point values of a set valued function $\bar{f}$, defined as

$$\bar{f}(x) = [f_*(x), f^*(x)] \subset \mathbb{R}, \quad \forall x \in \overline{X} \subset \mathbb{R}^n, \tag{1}$$

where f_* and f^* are real functions defined over $\mathbb{R}^n$ and

$$f_*(x) \leq f(x) \leq f^*(x), \quad \forall x \in \overline{X} \subset \mathbb{R}^n. \tag{2}$$

If f_* and f^* are continuous functions defined over a box

$$\overline{X} = \left\{x \in \mathbb{R}^n : a_i \leq x_i \leq b_i, i = 1, \ldots, n\right\}, \tag{3}$$

then the sets of global optimizers ('min' or 'max' points) are non-empty. Now we reproduce the notion of optimality in problems with uncertainties as presented in [2, pp. 856–857]:

The global max problem. Find

$$s^* = \max\left\{f_*(x), x \in \overline{X}\right\}, \tag{4}$$

* Corresponding author.
E-mail addresses: miguel@mat.uned.es (M. Delgado Pineda), galperin.efim@uqam.ca (E.A. Galperin).

0898-1221/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2010.02.013

and the set

$$X^* = \left\{x \in \overline{X} \mid f_*(x) = s^*\right\}. \tag{5}$$

If X^* is a singleton, the problem is solved. Otherwise, find

$$s^0 = \max\left\{f^*(x), x \in X^*\right\}, \tag{6}$$

and the set

$$X^0 = \left\{x \in X^* \mid f^*(x) = s^0\right\}. \tag{7}$$

The full global max solution is given by the set X^0 of all globally maximizing points and the constant segment

$$\overline{f}^0(x) = [s^*, s^0], \quad \forall x \in X^0. \tag{8}$$

The global min problem. Find

$$s^* = \min\left\{f^*(x), x \in \overline{X}\right\}, \tag{9}$$

and the set

$$X^* = \left\{x \in \overline{X} \mid f^*(x) = s^*\right\}. \tag{10}$$

If X^* is a singleton, the problem is solved. Otherwise, find

$$s^0 = \min\left\{f_*(x), x \in X^*\right\}, \tag{11}$$

and the set

$$X^0 = \left\{x \in X^* \mid f_*(x) = s^0\right\}. \tag{12}$$

The full global min solution is given by the set X^0 of all globally minimizing points and the constant segment

$$\overline{f}^0(x) = [s^0, s^*], \quad \forall x \in X^0. \tag{13}$$

Here the bar means closure. The functions of the band in (1)–(2), $f_*(x)$ and $f^*(x)$, are supposed to be Lipschitz continuous over $\overline{X}$. The algorithm in Section 3 is given for Lipschitz continuous functions. A function g is called Lipschitz continuous over $\overline{X}$ if

$$\left|g(x) - g(x')\right| \le A\left\|x - x'\right\|, \quad \forall x, x' \in \overline{X}, A = const, \min A = \max\left\|\nabla g(x)\right\|, x \in \overline{X}. \tag{14}$$

The functions $f_*(x)$ and $f^*(x)$ are not supposed to be given as formulas in the code; they are assumed to be computable.

On the basis of Lemma 2 of [2], the code is developed for the min problem only, and to solve a max problem one has to replace $\overline{f}(x)$ by $-\overline{f}(x)$ and the resulting $[s^0, s^*]$ by $[-s^*, -s^0]$, which is done automatically by the algorithm. The code for the gamma algorithm operates in the unit cube $[0, 1]^n$ common to all problems. For other specifics concerning problems with equality constraints, visualization via plane projections and sections, and some other details, the reader is referred to [3,4]. The formal procedure of the sequential gamma algorithm for the closed cube in $\mathbb{R}^n$ is presented in [1,2].

2. Transformation to the unit cube

It is expedient to transform a box (3) into the unit cube $\overline{U}$, axes oriented, with the edge $c = 1$ and the vertex at the origin, $\overline{U} = [0, 1]^n$.

The linear transformation

$$x_i = a_i + (b_i - a_i)z_i, \quad i = 1, \ldots, n \tag{15}$$

converts the problem (9)–(10) into the problem

$$\inf g^*(z) = \inf f^*[a + (b - a)z], \quad z \in \overline{U} = \{z \in \mathbb{R}^n : 0 \le z \le 1\}, \tag{16}$$

where we used self-evident vector notation in (16). It is the problem (16) that runs in numerical iterations performed by the code, yielding, in the limit, the unique *global* (absolute) minimum value

$$s^* = \min g^*(z) = \min f^*(x) = \inf f^*(x), \quad z \in \overline{U}, x \in \overline{X}, \tag{17}$$

and the entire set U^* of all global minimizers: $g^*(U^*) = s^*$. However, the process cannot be extended up to the very limit. So after a finite number of iterations, a quasi-cubic set $\overline{U^*}$ will be obtained such that

$$\overline{U^*} = \{z \in \overline{U^*} : s^* \le g^*(z) \le s^* + \varepsilon\} \subset \overline{U}. \tag{18}$$

It is over the set $\overline{U^*}$ in (18) that the numerical iterations performed by the code run for the problem (11)–(12), yielding, after a finite number of additional iterations, an approximate minimum value

$$s^0 \cong \min g_*(z) = \min f_*(x) = \inf f_*(x), \quad z \in \overline{U^*} \subset \overline{U}, x \in \overline{X}, \tag{19}$$

and the quasi-cubic set $\overline{U^0}$ such that

$$\overline{U^0} = \{z \in \overline{U^0} : s^0 - \eta \le g_*(z) = s^0 + \gamma < s^* + \varepsilon\} \subset \overline{U^*}. \tag{20}$$

This set $\overline{U^0}$ is then transformed into original coordinates by (15) within the box (3) yielding an approximation $\overline{X^0}$ to the solution set X^0 in (12). To obtain a better solution, further iterations similar to (18)–(20) starting from the quasi-cubic set $\overline{U^0}$ of (20) are performed.

The input for the optimization block of the code is (n, f^*, f_*, a, b) of (2)–(3) for computing an approximation $\overline{X^0}$ to X^0 of (12) and to $[s^0, s^*]$ of (13). The set X^0 may present in the limit one or several points in $\overline{X}$, a countable set of points in $\overline{X}$, or a line, surface or manifold (continuum) within $\overline{X}$. Since the algorithm is iterative, it yields the approximation $\overline{X^0}$ and $[s^{0\eta}, s^{*\eta}]$ with a specified precision $\eta > 0$, such that

$$s^0 - \eta \le s^{0\eta} \le s^{*\eta} \le s^* + \eta, \qquad s^0 - \eta \le f_*(x) \le f^*(x) \le s^* + \eta, \quad \forall x \in \overline{X^0}. \tag{21}$$

For convergence theorems related to each stage separately,

$$\eta_m \to 0, \qquad s_m^{0\eta} \to s^0, \qquad s_m^{*\eta} \to s^*, \qquad X^0 = \bigcap_m \overline{X_m^0}, \tag{22}$$

as $m \to \infty$; see [5, pp. 13–15]. The output of the optimization block of the code is the constant segment $[s^{0\eta}, s^{*\eta}]$ and the quasi-rectangular set $\overline{X^0}$ represented as a graph on the screen of the computer if $n = 2$, or in plane sections and projections for $n > 2$, or as a table; see Sections 5 and 6.

3. The gamma algorithm

For a better understanding of the code, we reproduce here the gamma algorithm from [2, pp. 858–859]. The gamma algorithm is simple and can be constructed by applying the cubic algorithm to (9)–(10) and to (11)–(12) in alternating mode. For the unit cube $\overline{U} \subset \mathbb{R}^n$ (see (16)), with the grid point at the origin, the gamma algorithm (GA) can be described as follows.

Iteration 1 *with upper function* g^*. Take an integer $N \ge 2$ (the subdivision ratio) and partition $\overline{U}$ into N^n smaller identical subcubes $\overline{C_i}$ such that $\bigcup \overline{C_i} = \overline{U}$, $C_i \cap C_j = \emptyset$, empty for all $i \ne j$ (here C_i = *interior* $\overline{C_i}$). The edge of each $\overline{C_i}$ is $\frac{1}{N}$ and its diameter is $\frac{\sqrt{n}}{N}$. For certain A_{g^*} (the slope bound for $g^*(z)$ of (16)), define the deletion constant for subcubes $\overline{C_i}$ denoted as $\overline{C_i^1}$:

$$r_1 = A_{g^*} \frac{\sqrt{n}}{N}. \tag{23}$$

Shift the subcube $\overline{C_{i_0}^1}$ with the grid point $z_{i_0}^1 = 0$ at the origin to coincide with each $\overline{C_i^1}$, one by one. This will define the grid point $z_i^1 \in \overline{C_i^1}$ for every $\overline{C_i^1}$ (this procedure is called [5, p. 9] translated grid generation). Given N^n points z_i^1, compute all values $g^*(z_i^1)$ and calculate

$$s_1^* = \min g^*(z_i^1), \quad 1 \le i \le N^n. \tag{24}$$

The number s_1^* provides the first approximation to the global minimum value s^* in (17); for the procedure of the cubic algorithm (CA), it is called [5, p. 10] the comparison constant; see (26).

Considering (14) for the cost function $g^*(z)$ of (16), we conclude that the variation of $g^*(z)$ over each $\overline{C_i^1}$ is bounded by r_1:

$$Var_{z \in \overline{C_i^1}} g^*(z) = \max_{z, z \in \overline{C_i^1}} \left|g^*(z) - g^*(z')\right| \le A_{g^*} \max_{z, z \in \overline{C_i^1}} \left\|z - z'\right\| = A_{g^*} \frac{\sqrt{n}}{N} = r_1. \tag{25}$$

This means that every $\overline{C_i^1}$ for which

$$g^*(z_i^1) - s_1^* > r_1, \quad z_i^1 = \text{grid point of } \overline{C_i^1}, \tag{26}$$

does not contain global minimizers and should be discarded. Deleting all subcubes (26), we obtain a quasi-cubic set (not necessarily connected) of the remaining closed subcubes

$$\overline{K}_1 = \left\{z \in \overline{C_i^1} : g^*(z_i^1) - s_1^* \le r_1, 1 \le i \le N^n\right\} \tag{27}$$

which provides the first approximation for the set $\overline{U^*}$ of (18) and, after the transformation to original coordinates, for the set X^* of (10).

Further iterations with g^*. Partition each $\overline{C_i^1} \subset \overline{K}_1$ in the same way as $\overline{U}$. The second deletion constant is $r_2 = \frac{r_1}{N}$. Generate new grid points $\{z_i^2\}$ in the same way and repeat Iteration 1 replacing z_i^1, r_1, s_1^* by z_i^2, r_2, s_2^*, etc., and stop when $r_m = \frac{r_{m-1}}{N} = A_{g^*} \frac{\sqrt{n}}{N^m}$ is sufficiently small.

Lemma 3 in [1, pp. 949–950] gives the stopping rule, by the following result:

$$0 \le s^*_m - s^* \le r_m, \qquad -r_m \le g^*(z) - s^*_m \le 2r_m, \qquad g^*(z) - s^* \le 3r_m, \quad \forall z \in \overline{K}_m. \tag{28}$$

Now we see that if the last iteration is m, then the precision of attaining the global minimum within $\overline{K}_m$ (i.e., the maximum elevation over the unknown s^* within $\overline{K}_m$) is

$$\eta_m = 3r_m = 3A_{g^*}\frac{\sqrt{n}}{N^m} = 3A_{g^*}\sqrt{n}\varepsilon_m, \tag{29}$$

where $\varepsilon_m = \frac{1}{N^m}$ is the edge of the subcubes in $\overline{K}_m$.

Thus, we have the approximate value $s^* \cong s^*_m$, and the approximate set $\overline{U^*} \cong \overline{K}_m$ which is represented by the set of grid points $z^m_j \in \overline{K}_m$. Those points z^m_j can be converted into original coordinates x^m_j using (15), and the list of those x^m_j represents the set $\overline{X^*_m}$, an approximation to $\overline{X^*}$ of (10), consisting of small rectangles corresponding to the grid $\{x^m_j\}$:

$$\overline{X^*_m} = \bigcup_j \overline{X^*_j}, \quad \overline{X^*_j} = \left\{x = (x_1, \ldots, x_n) : \left(x^m_j\right)_i \le x_i \le \left(x^m_j\right)_i + \frac{b_i - a_i}{N^m}\right\}. \tag{30}$$

Iteration 2 with lower function g_*. Compute all values $g_*(z^m_j)$, $z^m_j \in \overline{K}_m \subset \overline{U^*}$, and calculate

$$s^0_m = \min g_*(z^m_j), \quad z^m_j \in \overline{K}_m \subset \overline{U^*}. \tag{31}$$

The number s^0_m provides the first approximation to the global minimum value s^0 in (11).

Considering (14) for the cost function $g_*(z)$ of (19), we conclude that the variation of $g_*(z)$ over each $\overline{C^m_j} \subset \overline{K}_m$ is bounded by $\widetilde{r}_m = \widetilde{A}_{g_*}\frac{\sqrt{n}}{N^m}$, where $\widetilde{A}_{g_*}$ is the slope bound for $g_*(z)$:

$$Var_{z\in\overline{C^m_j}} g_*(z) = \max_{z,z\in\overline{C^m_j}} \left|g_*(z) - g_*(z')\right| \le \widetilde{A}_{g_*} \max_{z,z\in\overline{C^m_j}} \left\|z - z'\right\| = \widetilde{A}_{g_*}\frac{\sqrt{n}}{N^m} = \widetilde{r}_m. \tag{32}$$

This means that every $\overline{C^m_j} \subset \overline{K}_m$ for which

$$g_*(z^m_j) - s^0_m > \widetilde{r}_m, \quad z^m_j = \text{grid point of } \overline{C^m_j}, \tag{33}$$

does not contain global minimizers and should be discarded.

Further iterations with g_*. Generate new grid points $\{z^{m+1}_j\}$ in the same way and repeat Iteration 2, replacing $z^m_j, \widetilde{r}_m, s^0_m$ by $z^{m+1}_j, \widetilde{r}_{m+1}, s^0_{m+1}$, etc., and stop when $\widetilde{r}_p = \frac{\widetilde{r}_{p-1}}{N} = \widetilde{A}_{g_*}\frac{\sqrt{n}}{N^p}$ is sufficiently small.

We obtain a quasi-cubic set (not necessarily connected) of remaining closed subcubes $\overline{K}_p$ which provides an approximation set for the set $\overline{U^0}$ of (20) and, after the transformation to original coordinates, for the set $\overline{X^0}$ of (12).

Thus, we have the approximate value $s^0 \cong s^0_p$ and the approximate set $\overline{U^0} \cong \overline{K}_p$ which is represented by the set of grid points $z^p_j \in \overline{K}_p$. These points z^p_j are automatically converted into original coordinates x^p_j using (15), and the list of these x^p_j represents the set $\overline{X^0}$ of (21) consisting of small rectangles corresponding to the grid $\{x^p_j\}$:

$$\overline{X^0} = \bigcup_j \overline{X^0_j}, \quad \overline{X^0_j} = \left\{x = (x_1, \ldots, x_n) : \left(x^p_j\right)_i \le x_i \le \left(x^p_j\right)_i + \frac{b_i - a_i}{N^m}\right\}. \tag{34}$$

The set $\overline{X^0}$ is plotted on the screen for $n \le 2$. If $n > 2$ then projections and section planes are plotted on coordinate planes.

Of course, other forms of printout can be used, if required; alternatively, an output table can be printed containing points x^p_i of the set $\overline{X^0}$; see Section 6 with the MAPLE code.

Remark 3.1. The values of the constants A_{g^*} in (23), and $\widetilde{A}_{g_*}$ in (32), are of critical importance. If the minimal Lipschitz constant L_{g^*} for the function $g^*(z)$ over $\overline{U}$ is known and in (23) we have $A_{g^*} \ge L_{g^*}$, then the algorithm does not eliminate global minimizers and solves the problem with full guarantee. However, if $A_{g^*} < L_{g^*}$ (which may be the case if L_{g^*} is unknown), then the algorithm may eliminate some or all global minimizers (for estimates see [5, pp. 69–74]. Since large A_{g^*} imply slow convergence and increased computer time, a reasonable compromise in a choice of A_{g^*} is expedient. Table 1 in [4] presents for certain values of A the slopes of $f^*(x)$ in (1) such that global minimizers at the bottom of those slopes cannot be deleted. This allows one to choose A so as to preserve global minimizers corresponding to not too sharp slopes.

The algorithm minimizes $g^*(z)$ of (16) and $g_*(z)$ of (19), not $f^*(x)$ and $f_*(x)$ of (1). Defining $g^*_i = \frac{\partial g^*}{\partial z_i}, f^*_i = \frac{\partial f^*}{\partial x_i}, c_i = b_i - a_i$, we get from (16)

$$\left\|\nabla g^*(z)\right\| = \left(\sum_1^n g^{*2}_i\right)^{1/2} = \left(\sum_1^n c^2_i f^{*2}_i\right)^{1/2}, \tag{35}$$

and if for $f^*(x)$ the minimal Lipschitz constant L over $\overline{X}$ is known, we have

$$L \min_i c_i \leq \max_{z \in \overline{U}} \left\| \nabla g^*(z) \right\| \leq L \max_i c_i. \tag{36}$$

Thus, for a reasonable A of (14), the corresponding A_{g^*} in (23) should be chosen from the inequality

$$A \min_i c_i \leq A_{g^*} \leq A \max_i c_i, \quad c_i = b_i - a_i. \tag{37}$$

The case is the same for $g_*(z)$. This uncertainty is inherent to the algorithm in the case of unknown or too high Lipschitz constants $\max \|\nabla g^*(z)\|$, $\max \|\nabla g_*(z)\|$, for $z \in \overline{U}$.

4. Specific features incorporated in the code

The code has modular structure. Such structure ensures reliability and flexibility of the code. An interested user can change a block according to his/her special needs or add another block to solve a different problem via global minimization using the main block as a tool.

The code is written in MAPLE in the user–machine interface mode. This mode is convenient for applications as well as for research and educational purposes. A function (such as a formula or a procedure), input data and the necessary min/max specification are typed directly into the text of the code on the screen of computer. Then, at the push of a key, the code solves the problem and prints out a table of solution data or a graph, as required. The user can make experiments on-line, changing parameters of the model (i.e. in the formula or the procedure), or the model itself, or some or all input data, monitoring the effects visually on the screen. The functions should be written as formulas or procedures representable in MAPLE within the text of the code using the keyboard.

The following specific features are incorporated in the code.

1. *Slope control*. The user can change the constant A (slope bound) in the code according to his/her knowledge of the slope that the extreme functions in (2) may have. The exact bound is defined by the smallest Lipschitz constant L, but it is usually unknown and difficult to determine. With a greater A, the computation time increases; in contrast, a smaller A speeds up the computation. However, if $A < L$, then determination of all global minimizers is not guaranteed: some or all of them may be lost. In a case of doubt, one needs to increase A in order to check whether new minimizers would appear.
2. *Scale control*. Another way to modify the slope without affecting the set of global minimizers is to multiply the bounding functions of the cost function by a small constant. For example, polynomials of high degree ($k \geq 5$) usually have peaks with high slopes. To avoid wasting computer time for large A, or loss of roots for $A < L$, it is expedient, after the transformation to the unit segment, $z \in [0, 1]$, of the region $x \in [a, b]$ where the roots are sought, to divide the transformed polynomial by the greatest absolute value of its coefficient. This operation is automatic in the code, but it may lead to loss of precision due to small values appearing in the computation. In such a case, manual intervention of the user is possible, to modify the multiplier.
3. *Comparison constant control*. In the code, the choice of the comparison constants $s^0_{m,}$ and s^*_m is automatic, or simply $s_m = 0$, $\forall m$. However, to speed up the computation, or to look for some specific minimizers, the user may wish to set $s_m = 0$ for some iterations, or $s_m > 0$ for some problems if the existence of roots $g(z) = 0$ in a given rectangle is in doubt. In such a case, the global min value $s^0 > 0$ presents a measure for the lack of knowledge over that rectangle.
4. *Time check and suboptimal solutions*. If allotted computer time is running out or if an acceptable suboptimal solution is already obtained, the iterations can be terminated at will, switching to the output of the current iteration.
5. *Intermediate outputs*. At any iteration, the user can call the output block to see the results of that iteration, and then continue, modify or terminate the procedure. This is very useful in computational experiments and may save time in lengthy computations.
6. *A cleaning procedure*, to avoid accumulation of ε-close solution points; see Section 6.

5. Use of the code and numerical experiments

A numerical example is presented in the text of the code to illustrate how to use it and where to type the initial information into the text of the code. The user should simply retype in the same spaces the input information of his/her own problem.

A user proficient in MAPLE can easily use the code and modify it if needed. Detailed comments are included in the program which begins with the main block A of basic procedures needed for application of the cubic algorithm [5, pp. 7–24]. There then follows block B for solution of concrete problems with a solved example intended to help the user to learn the program and to use it for solving his/her own problems by substituting a new function with new data.

The example below illustrates the current market meltdown in the US and global economy, that falls nicely in line with Hyman Minsky's (1919–1996) Financial Instability Hypothesis [6]. At different times and intervals, we see zigzag wave curves showing increases and decreases of stock prices and different economic and financial indices. Such evolution can be modelled via a choice of sine and cosine functions adjusted to the past rise and current fall of the economy which took place

Table 1
Minimization problem (41)–(42).

$\varepsilon = 0.001$
Number_of_Iterations = 15
Edge_of_smaller_subcube = 0.00048828125
Number_of_subcubes_remaining = 66
Number_of_grid_points_epsilon-separated = 3
Uncertainty_band $[s^0, s^*] = [0.0538619196, 0.3237633902]$
Minimizers_set = {[1.9999, 1.9999]} ± {[0.000488, 0.000488]}, the ε-approximation to X^0

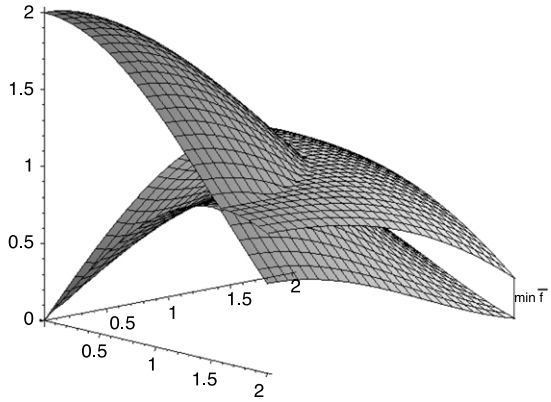

Fig. 1. Graph of the functions (41) and (42).

over the last couple of years, 2006–2008. Those functions in (41)–(42) below may be substituted by the real data curves over a certain time interval to study the evolution of a sector in the market economy or finance. For simplicity in illustrating the gamma algorithm, we prefer to use the representation based on suitable formulas that reflect the rise and fall in the evolution of economic realities.

Block B, *Global optimization*, contains a solved problem in two variables:

$$\min \bar{f}(x_1, x_2) \tag{38}$$

$$x_1 \in [0, 2], \qquad x_2 \in [0, 2]. \tag{39}$$

$$\bar{f}(x_1, x_2) = [f_*(x_1, x_2), f^*(x_1, x_2)] \tag{40}$$

where

$$f_*(x_1, x_2) = \min\left(\left|\sin\sqrt{(x_1)^2 + (x_2)^2}\right|, \left|1 + \cos\sqrt{(x_1)^2 + (x_2)^2}\right|\right), \tag{41}$$

$$f^*(x_1, x_2) = \max\left(\left|\sin\sqrt{(x_1)^2 + (x_2)^2}\right|, \left|1 + \cos\sqrt{(x_1)^2 + (x_2)^2}\right|\right). \tag{42}$$

This problem has the global min solution as the constant segment $[s^0, s^*] = [0.0538619196, 0.3237633902]$ and a singleton from the global minimizers $X^0 = \{(2, 2)\}$ (the solution set; see Table 1), which are plotted as a graph in the text of the code (see Fig. 1).

Changing the precision $\varepsilon > 0$, the user can play with it to see changing level sets that shrink into a singleton as $\varepsilon \to 0$. Table 1 shows the output.

6. Maple code in R^n of the gamma algorithm for global optimization of uncertain functions with box constraints

```
>restart:with(plots):with(LinearAlgebra): with(linalg):
Division constant N>=2 (user's choices)
>N:=2:
A. Listing of procedures employed
A1. Transformation to the unit cube. Transformation of an array of points from the unit cube to the given box
>pointR:=proc(Point,Rectangle) local i,a,b,NewPoint:
>NewPoint:=[]:
>for i from 1 to nops(Point) do
>a:=op(1,Rectangle[i]):b:=op(2,Rectangle[i]):NewPoint:=[op(NewPoint),(b-a)*Point[i]+a]:od:
>RETURN(NewPoint)end:
```

```
>ListPointsR:=proc(listPoints,Rectangle) local NewList,i:
>NewList:=[]:
>for i from 1 to nops(listPoints) do
>NewList:=[op(NewList),pointR(listPoints[i],Rectangle)]:od:
  >RETURN(NewList):end:
```

A2. *Initialization: coordinate of the origin*

```
>PointOrigin:=proc(Dimension) local Origin, i:
>Origin:=[]:for i from 1 to Dimension do
>Origin:=[op(Origin),0]:od:
  >RETURN(Origin)end:
```

Initialization: vertices of the unit n-cube; (binary array)

```
>listBinary:=proc(Dimension) local nbinary, i:
>nbinary:=[]:
>for i from 0 to 2^Dimension-1 do nbinary:=[op(nbinary),convert(i,binary)]:od:
  >RETURN(nbinary):end:
```

Vertices

```
>vertexCubeU:=proc(Dimension) local lpoints, nnbb, number, vectnumber, i, j, long:
lpoints:=[]:nnbb:=listBinary(Dimension):
>for j from 1 to nops(nnbb) do
>number:=nnbb[j];long:=length(number);vectnumber:=[];
>for i from Dimension to 1 by -1 do
>if long<i then vectnumber:=[op(vectnumber),0]
else vectnumber:=[op(vectnumber),floor(number/10^(i-1))]:
number:=number-floor(number/10^(i-1))*10^(i-1): fi: od:
>lpoints:=[op(lpoints),vectnumber]:od:
  >RETURN(lpoints):end:
```

A3. *Translated grid generator* [5, p. 9]

```
>Ncube:= proc(Point,Valueh,Dimension) local listavertex,j,lpoints; listavertex:=[]:
lpoints:=vertexCubeU(Dimension):
for j from 1 to nops(lpoints) do listavertex:=[op(listavertex),Point+jump(Valueh)*lpoints[j]]: od:
  >RETURN(listavertex); end:
```

A4 *Point purification: comparison constant generator* [5, p. 10] *and deletion operator* [5, pp. 10–11]

```
>DepurePoints:= proc(ListPoints,IVE,Valueextreme,ValueA,Valueh,Dimension)
>local TableValues, mTableValues, NewList, decide, aproxdecide, i:
>TableValues:=[]:
>for i from 1 to nops(ListPoints) do
>TableValues:=[op(TableValues),g(ListPoints[i])]
>od:
>if (IVE=1) then mTableValues:=Valueextreme; else mTableValues:=min(op(TableValues));end if:
>NewList:=[]:
>for i from 1 to nops(ListPoints) do
>decide:=TableValues[i]-mTableValues-ValueA*diameter(Valueh,Dimension):
aproxdecide:=evalf(decide):
>if aproxdecide<=0 then NewList:=[op(NewList),ListPoints[i]]:fi:
>od;
  >RETURN(NewList):end:
```

A5. *Procedure to generate vertices for the next iteration* [5, p. 12]

```
>CutCubes:=proc(ListPoints,Valueh,Dimension) local NewList,i:
>NewList:=[]:
>for i from 1 to nops(ListPoints) do
>NewList:=[op(NewList),op(Ncube(ListPoints[i],Valueh,Dimension))]:
>od:
  >RETURN(NewList):end:
```

A6. *Cleaning procedure to eliminate too close points*

```
>ClearPoints:= proc(ListPoints) local LDifference, MDifference, NewList, i, k, value, tope:
NewList:=[ListPoints[1]]; tope:=nops(ListPoints):
>for i from 1 to tope do
>value:=0:
>for k from 1 to nops(NewList) do
>LDifference:=ListPoints[i]-NewList[k]:
MDifference:=max(abs(max(op(LDifference))),abs(max(op(LDifference)))):
```

```
>if MDifference>=10*epsilon1 then value:=value+1: fi:
>od:
>if value=nops(NewList) then NewList:=[op(NewList),ListPoints[i]]; fi;
>od:
  >RETURN(NewList):end:
```

A7. *Subdivision of subcubes*

```
>edge:=1:jump:=t->edge/t:
>diameter:=proc(t,Dimension) RETURN(edge*Dimension^(1/2)/t) end:
```

Transformation of point coordinates into graphic representation

```
>sameTwo:=proc(listvalues) local NewList,i, value:
>NewList:=[]:
>for i from 1 to nops(listvalues) do
>value:= [op(1,listvalues[i])]:
NewList:=[op(NewList),[op(value),0],[op(value),f1(value)],[op(value),f2(value)]]:od:
  >RETURN(NewList):end proc:
```

A8. *Projections and Sections by planes*

Projection onto coordinate planes

```
>proyection:=proc(ListPointsRn,listXY) local NewList,i,a,b:
>NewList:=[]:
>for i from 1 to nops(ListPointsRn) do
>a:=op(1,listXY):b:=op(2,listXY):
NewList:=[op(NewList),[op(a,ListPointsRn[i]),op(b,ListPointsRn[i])]]:od:
  >RETURN(NewList):end:
```

Section by a plane

```
>planesection:=proc(ListPointsRn,listVector) local NewList,i,a,b,M,valorM:
>NewList:=[]:
>for i from 1 to nops(ListPointsRn) do
>M := Matrix([op(listVector),ListPointsRn[i]]):valorM:=abs(Determinant(M)):
>if epsilon>=valorM then NewList:=[op(NewList),ListPointsRn[i]]: fi:od:
  >RETURN(NewList):end:
```

Projection onto a plane

```
>planeproyection:=proc(ListPointsRn,listVector) local NewList,i,a,b,j,M, Newpoint:
>NewList:=[]:
>for i from 1 to nops(ListPointsRn) do
>Newpoint:=[]:
>for j from 1 to nops(listVector) do
>M := dotprod(vector(op(j,listVector)),vector(ListPointsRn[i])):
Newpoint:=[op(Newpoint),M]: od:
>NewList:=[op(NewList),Newpoint]:od:
  >RETURN(NewList):end:
```

Combination(n,2)

```
>combination:=proc(n) local NewList,i,j:
>NewList:=[]:
>for i from 1 to n-1 do
>for j from i+1 to n do NewList:=[op(NewList),[i,j]]:od:
>od:
  >RETURN(NewList):end:
```

A9. *Iterative Procedures of the Cubic Algorithm*

A9.1 *Cubic Algorithm* [5, pp. 11–13]. *This procedure works with the initial box*

```
>CubicAlgorithm:=proc(dimension)
>local Vertex,h,m,ListPoints,MinPoints,GlobalMinPoints:
>global HRectangle:
>h:=1:m:=0:Vertex:=PointOrigin(dimension):
>h:=N*h:m:=m+1:ListPoints:=Ncube(Vertex,h,dimension):
```

Minimizers (Maple list) (until that edge_subcube<epsilon1)

```
>while jump(h)>epsilon1 do
>ListPoints:=DepurePoints(ListPoints,ive,d*ValueExtreme,A,h,dimension):
>MinPoints:=ListPoints:MinPoints;
>h:=N*h:m:=m+1:ListPoints:=CutCubes(ListPoints,h,dimension):od:
```

Approximated set of the minimizers set (Maple list) (epsilon-separated)

```
>if nops(ListPoints)=0 then GlobalMinPoints:=[]:
```

```
   else GlobalMinPoints:=ClearPoints(MinPoints):fi:
      >RETURN([MinPoints,GlobalMinPoints,h,m]):end:
```

A9.2.*Cubic Algorithm2* [5, pp. 11–13]. *This procedure works with quasi-rectangular set*

```
   >CubicAlgorithm2:=proc(dimension,MinPointsStar,GlobalMinPointsStar,hStar,mStar)
   >local Vertex,h,m,ListPoints,MinPoints,GlobalMinPoints:
   >global HRectangle:
   >h:=hStar/N:m:=mStar-1:ListPoints:=MinPointsStar:
```

Minimizers (Maple list) (until that edge_subcube<epsilon1)

```
   >while jump(h)>epsilon1 do
   >ListPoints:=DepurePoints(ListPoints,ive,d*ValueExtreme,A,h,dimension):
   >MinPoints:=ListPoints:MinPoints;
   >h:=N*h:m:=m+1:ListPoints:=CutCubes(ListPoints,h,dimension):od:
```

Approximated set of minimizers set (Maple list) (epsilon-separated)

```
   >if nops(ListPoints)=0 then GlobalMinPoints:=[]:
   else GlobalMinPoints:=ClearPoints(MinPoints):fi:
      >RETURN([MinPoints,GlobalMinPoints,h,m]):end:
```

A11. *Output procedure*

```
   >outputData:=proc(dimension)
   >local PP1, PP2, PP1Dos, PP2Dos, PP, PPP1, PPP2, PPP, x1, x2, y1, y2, i, listproy,
   listaux, gtitle,puntos1, puntos2, puntos, sectiontitle, projectiontitle, listplot1, cosa, cosa1,
   cosa2, high:
   >if (pne=1)then PP2:=ListPointsR(MinPoints,HRectangle):
   else PP2:=ListPointsR(GlobalMinPoints,HRectangle):end if:
   >PP1:=ListPointsR(LPointsVertices,HRectangle):
   >if (dimension=1) then PP1Dos:=sameTwo(PP1):PP2Dos:=sameTwo(PP2):
   PP:=[PP1Dos,PP2Dos]:
   x1:=op(1,HRectangle[1]):x2:=op(2,HRectangle[1]):y1:=op(1,HRectangle[1]):y2:=op(2,HRectangle[1]):
   end if:
   >if (dimension=2) then PP:=[PP1,PP2]: x1:=op(1,HRectangle[1]):x2:=op(2,HRectangle[1]):
   y1:=op(1,HRectangle[2]):y2:=op(2,HRectangle[2]):end if:
   >print(op(1,message),op(2,message),op(3,message),op(4,message));
   >print(Number_of_Iterations=m);print(Edge_of_smaller_subcube=evalf(jump(h)));
   print(Number_of_subcubes_remaining=nops(MinPoints));
   >print(Number_of_grid_points_epsilon-separated=nops(GlobalMinPoints));
```

Output of the global min or max value of the function

```
   >if nops(GlobalMinPoints)=0 then print("Problem without solution");
   else if (smM=1)and (nops(GlobalMinPoints)<>0)then print(Value_extreme=evalf([minimo2,minimo1]));fi;fi;
```

Tables of the numerical output of the solution

```
   >if (sn=1)and(ve=1) then print(PP2);fi;
   >if (sn=1)and(ve<>1) then print(evalf(PP2));fi;
```

Graphs of the set of global optimizers $R^{\mathbf{n}}$, n = 1, 2, 3

```
   >if (sg=1)and(dimension<3) then
   >print(plot(PP,x=x1..x2,y=y1..y2,axes=BOXED,scaling=CONSTRAINED,color=[black,red],
   style=point,symbol=[POINT,CROSS],title=op(1,message)));fi;
   >if (sg=1)and(dimension=2) then
   >listplot1:=[]:
   >for i from 1 to nops(PP1) do:
   >cosa:=op(i,PP1):cosa1:=[op(cosa),y1]:cosa2:=[op(cosa),y2]:
   >listplot1:=[op(listplot1),polygonplot3d([cosa1,cosa2],axes=boxed)]:
   >end do:
   >high:=max(epsilon,0.01):for i from 1 to nops(PP2) do:
   >cosa:=op(i,PP2):cosa1:=[op(cosa),minimo2]:cosa2:=[op(cosa),minimo1+high]:
   >listplot1:=[op(listplot1),polygonplot3d([cosa1,cosa2],axes=boxed)]:
   >end do:
   >print(display(listplot1));
   >fi;
   >if (sg=1)and(dimension=3) then
   >listaux:=PP1: puntos1:={seq(listaux[i],i=1..nops(listaux))}:
   >listaux:=PP2: puntos2:={seq(listaux[i],i=1..nops(listaux))}:puntos:=‘union‘(puntos1,puntos2):
   print(pointplot3d(puntos, color=red,axes=BOX,symbol=POINT));fi;
```

Projection onto coordinate planes for n > 2

```
>if (sg=1)and(ss=1)and(dimension>2) then
>print(op(1,message),op(2,message),op(3,message),op(4,message)):
>listproy:=combination(dimension):
>for i from 1 to nops(listproy) do
>PPP1:=proyection(PP1,listproy[i]):PPP2:=proyection(PP2,listproy[i]):PPP:=[PPP1,PPP2]:
>x1:=op(1,HRectangle[i]):x2:=op(2,HRectangle[i]):y1:=op(1,HRectangle[i]):y2:=op(2,HRectangle[i]):
>gtitle:=cat("Proyección plano X[",op(1,listproy[i]),"]X[",op(2,listproy[i]),"]"):
>print(plot(PPP,x=x1..x2,y=y1..y2,axes=BOX,scaling=CONSTRAINED,color=[red,blue],
style=point,symbol=[POINT,CROSS],title=gtitle));
>od:fi;
```

Section and Projection onto a plane for $n = 3$

```
if (sg=1)and(ss=0)and(dimension=3) then
listaux:=planesection(listMinPoints[1], op(1,listPlanes)):puntos1:={seq(listaux[i],i=1..nops(listaux))}:
listaux:=planesection(listMinPoints[2], op(1,listPlanes)):puntos2:={seq(listaux[i],i=1..nops(listaux))}:
puntos:=`union`(puntos1,puntos2):
print(pointplot3d(puntos, color=red,axes=BOX));fi;
```

Graphical representation of the function of one or two variables

```
>if dimension=1 then print(plot({f1([x,y]),f2([x,y])},x=x1..x2,y=y1..y2,axes=normal));fi;
>if dimension=2 then print(plot3d({f1([x,y]),f2([x,y])},x=x1..x2,y=y1..y2,axes=normal));fi;
>end proc:
```

A12. *Output Section procedure*

```
>outputSectionData:=proc(dimension,listvectors) local PP1, PP2, PP1Dos, PP2Dos, PP, PPP1, PPP2,
PPP,x1,x2,y1,y2,i,listproy,listaux, gtitle,puntos13D, puntos23D, puntos3D,puntos12D,sectiontitle,
projectiontitle, puntos22D, puntos2D,sectiontitle,projectiontitle:
>if (pne=1)then PP2:=ListPointsR(MinPoints,HRectangle):
else PP2:=ListPointsR(GlobalMinPoints,HRectangle):end if:
PP1:=ListPointsR(LPointsVertices,HRectangle):
>sectiontitle:=cat("3D View of Section for plane, Origin and ",convert(listvectors,string)):
>projectiontitle:=cat("2D View of section on plane, Origin and ",convert(listvectors,string)):
```

Section and Projection onto a plane for n=3:

```
>if (sg=1)and(dimension=3) then
>listaux:=planesection(PP1,op(1,listPlanes)):puntos13D:={seq(listaux[i],i=1..nops(listaux))}:
listaux:=planeproyection(PP1,listvectors):puntos12D:={seq(listaux[i],i=1..nops(listaux))}:
>listaux:=planesection(PP2,op(1,listPlanes)):puntos23D:={seq(listaux[i],i=1..nops(listaux))}:
listaux:=planeproyection(PP2,listvectors):puntos22D:={seq(listaux[i],i=1..nops(listaux))}:
>puntos3D:=`union`(puntos13D,puntos23D):
>puntos2D:=`union`(puntos12D,puntos22D):
>print(pointplot3d(puntos3D,symbol=point,color=green,axes=BOX,title=sectiontitle));
>print(pointplot(puntos2D,symbol=point,color=green,axes=BOX,title=projectiontitle));fi;
>end proc:
```

A13. *Iterative Procedure of the Gamma Algorithm; Gamma AlgorithmR* [2, p. 856]:

```
>GammaAlgorithmR:=proc(dimension)
>local minimo1,minimo2,Lipschitz,Holder, i, h, h0, m, m01, m001, m1, m02, m002, m2, valor1,
valor2, listVE1, listVE2, listm1, listm2, newliststackData, LPointsVertices, listData, ListPoints0,
ListPoints1, ListPoints, MinPoints, GlobalMinPoints, stackData, MPoints, GMinPoints,
listMinPoints, listGlobalMinPoints, liststackData, lista, listb, listh, listm, g1, g2:
>global SetX,HRectangle,f,g,f1,f2:
```

Computation of approximate global solutions by calling procedures from main Block A9

```
>listMinPoints:=[]:listGlobalMinPoints:=[]:listh:=[]:listm:=[]:liststackData:=[]:
listData:=[]:listVE1:=[]:listVE2:=[]:
>LPointsVertices:=vertexCubeU(n):
>listData:=[]:
>f:=proc(x) RETURN(f1(x)): end proc:
>g:=proc(x) RETURN(d*CB*f(pointR(x,HRectangle))) end proc:
g1:=proc(x) RETURN(d*CB*f(pointR(x,HRectangle))) end proc:
>print(f(x));
>stackData:=CubicAlgorithm(n):listData:=[op(listData),stackData]:cosa1:=stackData:
>MPoints:=op(1,stackData):
>GMinPoints:=op(2,stackData):
>h:=op(3,stackData):
>m:=op(4,stackData):print(Value_extreme=evalf(g(op(1,MPoints))));
```

```
>f:=proc(x) RETURN(f2(x)): end proc:
>g:=proc(x) RETURN(d*CB*f(pointR(x,HRectangle))) end proc:
g2:=proc(x) RETURN(d*CB*f(pointR(x,HRectangle))) end proc:
>print(f(x));
>stackData:=CubicAlgorithm2(n,MPoints,GMinPoints,h,m):listData:=[op(listData),stackData]:
>MPoints:=op(1,stackData):
>GMinPoints:=op(2,stackData):h:=op(3,stackData):m:=op(4,stackData):
>print(Value_extreme=evalf(g(op(1,MPoints))));
>liststackData:=[op(liststackData),listData]:
```

Select points of extremum values

```
>if (ive=1) then MinPoints:=MPoints:GlobalMinPoints:=GMinPoints:minimo1:=ValueExtreme:
minimo2:=ValueExtreme: else
>MinPoints:=[]:GlobalMinPoints:=[]:listVE1:=[]:listVE2:=[]:listm1:=[]:listm2:=[]:listh:=[]:
>for i from 1 to nops(liststackData) do:
>valor1:=op(1,op(1,op(1,op(i,liststackData)))):listVE1:=[op(listVE1),g1(valor1)]:
>listm1:=[op(listm1),op(4,op(1,op(i,liststackData)))]:
listm2:=[op(listm2),op(4,op(2,op(i,liststackData)))]:
>listh:=[op(listh),op(3,op(1,op(i,liststackData))),op(3,op(2,op(i,liststackData)))]:
>end do:
>minimo1:=min(op(listVE1)):h:=max(op(listh)):m:=max(op(listm2)):
>newliststackData:=[]:
>for i from 1 to nops(liststackData) do:
>valor1:=op(1,op(1,op(1,op(i,liststackData)))):m01:=g1(valor1):m001:=evalf(minimo1-m01):
m1:=evalf(abs(m001)-epsilon):
>if (m1<0) then newliststackData:=[op(newliststackData),op(2,op(i,liststackData))]:end if:
>end do:
>for i from 1 to nops(newliststackData) do:
>valor2:=op(1,op(1,op(i,newliststackData))):listVE2:=[op(listVE2),g2(valor2)]:
>end do:minimo2:=min(op(listVE2)):
>for i from 1 to nops(newliststackData) do:
>valor2:=op(1,op(1,op(i,newliststackData))):m02:=g2(valor2);m002:=evalf(minimo2-m02):
m2:=evalf(abs(m002)-epsilon):
>lista:=op(1,op(i,newliststackData)):listb:=op(2,op(i,newliststackData)):
>if (m2<0) then MinPoints:=[op(MinPoints),op(lista)]:
GlobalMinPoints:=[op(GlobalMinPoints),op(listb)]: end if:
>end do:end if:
   >RETURN([MinPoints,GlobalMinPoints,h,m,minimo1,minimo2,1,cosa1]):end proc:
```

B *Global optimization of uncertain functions with Gamma Algorithm*

B1. *Input (into the text of the code)*

Minimize (d=1) or maximize (d=-1)

```
>d:=1:
```

sm=0 (ive=1) or no (ive=0)

```
>ive:=1: ValueExtreme:=0:
```

Approximation error in function value space

```
>epsilon:=1/10 ^3:
```

Space dimension $R^{\mathbf{n}}$

```
>n:=2:
```

Hyper-rectangle (box) of constrains [a,b]= [[a $_1$*,b* $_1$*],[a* $_2$*,b* $_2$*], …,[a* $_n$ *b* $_n$*]]*

```
>HRectangle:=[[0,2],[0,2]]:
```

Function to optimize, a point x= (x[1],x[2], …,x[n])

```
>fD:=proc(x) RETURN(abs(sin(((x[1])^2+(x[2])^2)^(1/2))))end proc:
(Example2: fD:=proc(x) RETURN(0*x[1])end proc:)
>fU:=proc(x) RETURN(abs(1+cos(((x[1])^2+(x[2])^2)^(1/2))))end:
(Example2: fU:=proc(x) RETURN(abs(sin(0.5+9.5((x[1])^2+(x[2])^2)^(1/2))))end proc:)
```

Slope bound (user's choice, recommended A>Lip const)

```
>A:=1:
```

Scale multiplier (intermediate slope control, CB= 1, 0.1, …0.01)

```
>CB:=0.1:
```

B2. *Output procedure*

Points non eliminated (pne=1) or points epsilon-separated (pne=0) (user's choice)

```
>pne:=1:
```

Graphics: Yes (sg=1) o no (sg=0) (user's choice)
>sg:=1:
Output Projection plane: (ss=1) or Section plane(ss=0):
ss:=0:
List of planes to project:
listPlanes:=[[[1,1,0],[0,0,1]]]:
listPlanes:=[[[1,1,0],[0,0,1]],[[1,2,0],[0,0,1]]]:
Numerics: Yes (sn=1) o no (sn=0) Table of solution set (user's choice)
>sn:=0: ve:=1:
Yes (smM=1) or NO (smM=0) shows the extremum considered: (user's choices)
>smM:=1:
B3. *Data control*
Transformation of value epsilon for the unit cube.
>newlist:=[]:for i from 1 to nops(HRectangle) do:
>newlist:=[op(newlist),(op(2,HRectangle[i])-op(1,HRectangle[i]))]:end do:
>epsilon1:=epsilon/max(op(newlist)):
Message with problem, function and set
>punto:=[]:for i from 1 to 6 do punto:=[op(punto),x[i]]: end do:
Kind of problem control
>if (d=1) then d:=1: message:=["Minimization problem ", [f2(x),f1(x)]," in the set ",HRectangle]:
else d:=-1:message:=["Maximization problem ",[f2(x),f1(x)]," in the set ",HRectangle]: end if:
Used functions (auxiliary functions)
>f1:=proc(x) RETURN(max(fU(x),fD(x))): end proc:
>f2:=proc(x) RETURN(min(fU(x),fD(x))): end proc:
Dimension control
>if (abs(floor(n))=0) then n:=1 else n:=abs(floor(n)) end if:
Hyper-rectangle control
>if (n<>nops(HRectangle)) then Rectangle_bad_defined end if;
Numeric output control
>if (sn=1) then sn:=1 else sn:=0 end if:
Kind Control
>if (smM=1) then smM:=1 else smM:=0 end if:
Edge unit cube
>edge:=1:
Edge circumscribed cube
>edgeCube:=abs(op(1,op(1,HRectangle))-op(2,op(1,HRectangle))):
B4. *Computation of approximate global solutions by calling procedures from main Block A 9.*
>LPointsVertices:=vertexCubeU(n):
>stackData:=GammaAlgorithmR(n):
>MinPoints:=op(1,stackData):
>GlobalMinPoints:=op(2,stackData):
>h:=op(3,stackData):m:=op(4,stackData):
>minimo1:=op(5,stackData):
>minimo2:=op(6,stackData):
B5. *CubicOutput of data*
>outputData(n):
B6. *Output Projection and Section of data*
listPlanes:=[[[1,0,0],[0,0,1]]]:outputSectionData(n,listPlanes[1]);
listPlanes:=[[[1,0,0],[0,1,0]]]:outputSectionData(n,listPlanes[1]);
listPlanes:=[[[0,0,1],[0,1,0]]]:outputSectionData(n,listPlanes[1]);
listPlanes:=[[[0,1,0],[evalf(1/sqrt(2)),0,evalf(1/sqrt(2))]]]:outputSectionData(n,listPlanes[1]);
listPlanes:=[[[1,0,0],[0,evalf(1/sqrt(2)),evalf(1/sqrt(2))]]]:outputSectionData(n,listPlanes[1]);
listPlanes:=[[[0,0,1],[evalf(1/sqrt(2)),evalf(1/sqrt(2)),0]]]:outputSectionData(n,listPlanes[1]);

Acknowledgements

The research of the first author was partially supported by the grant S-0505/TIC/0230 from the Comunidad Autónoma de Madrid and SEI2006-15401-C04-02 from DGI of the Spanish Ministerio de Ciencia y Tecnología.

References

[1] E.A. Galperin, Global optimization in problems with uncertainties, Journal of Nonlinear Analysis 47 (2001) 941–952.
[2] E.A. Galperin, Global optimization in problems with uncertainties: The gamma algorithm, Computer and Mathematics with Applications 44 (2002) 853–862.
[3] M. Delgado Pineda, E.A. Galperin, Global optimization in R^n with box constraints and applications: A *Maple* code, Mathematical and Computer Modelling 38 (2003) 77–97.
[4] M. Delgado Pineda, E.A. Galperin, Global optimization over general compact sets by the beta algorithm: A *Maple* code, Computer and Mathematics with Applications 52 (2006) 33–54.
[5] E.A. Galperin, The Cubic Algorithm for Optimization and Control, NP Research Publ., Montreal, 1990.
[6] R.L. Wray, Financial markets meltdown: What can we learn from Minsky? Public policy Brief no. 94, Levy Economics Institute of Bard College, New York, 2008.